Customized XRootD Client Library

for Gluex raw data production on the OSG

Richard Jones, University of Connecticut
and
the GlueX Collaboration

Gluex use case for xrootd: *raw data production*

Established process

- **JANA** (JLab ANAlysis) framework
  - core multi-threaded application - *hd_root*
  - producer/consumer model
  - data sources (readers), sinks (writers)
  - consumers are plugins -- *many*
  - standard production: 29 plugins

- **EVIO** (Jlab) raw data format
  - variable-size blocks ~60MB, 1k events
  - sequential access at block level
  - random access within blocks
  - file-based (not streaming)
  - special blocks at file beginning, end

Gluex use case for xrootd: *raw data production*

Established process

- **JANA** (JLab ANAlysis) framework
  - core multi-threaded application - *hd_root*
  - producer/consumer model
  - data sources (readers), sinks (writers)
  - consumers are plugins -- *many*
  - standard production: 29 plugins

- **EVIO** (Jlab) raw data format
  - variable-size blocks ~60MB, 1k events
  - sequential access at block level
  - random access within blocks
  - file-based (not streaming)
  - special blocks at file beginning, end

Established workflow

- **HPC production** resource model
  - assumes many-core resource
  - runs whole-node on JLab farm
  - now running on offsite HPC sites
    - NERSC
    - PSC
    - IU

- **SWIFT-2** (JLab) workflow managment
  - tools to manage distribution of data to individual sites
  - globus-online for data movement

Gluex use case for xrootd: *raw data production*

What is the scale of the resources needed?

- experimental event rate = 75kHz @ 1.4kB / event ~ 1GB/s, 3-4 PB / year
- raw data processing rate (2.4GHz EPYC) = 10k events / core-hr
- steady-state production = 65M core-hr / year

Where are we now?

- in a shutdown (JLab SAD)
- nearly caught up, but have encountered bottlenecks with off-site HPC
- ***limited by current allocations, difficult to contemplate second passes...***
- ***Gluex OSG resources*** (campus grids, ComputeCanada) dedicated to simulations
- " " " ***are underused -- can these be used for raw data?***

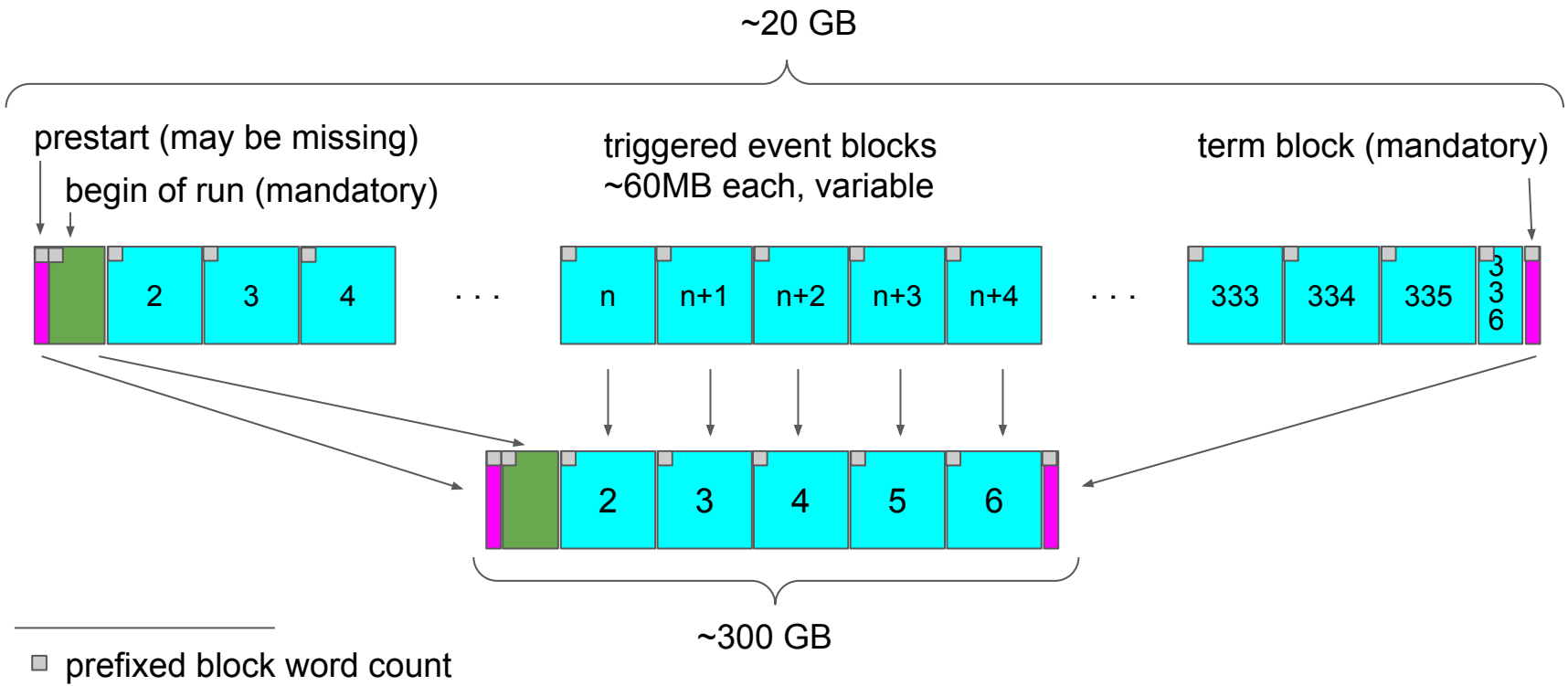
Gluex challenge: *raw data production on HTC*

Challenge: port an application designed for HPC to run on HTC

1. ***Split*** up the input data into schedulable chunks (2 core-hr)
   - 20GB evio file = 1.4M events = 130 core-hr
   - 340 evio blocks / file
   - 5 evio blocks / job
   - 70 jobs / evio file

2. ***Merge*** results after processing
   - much smaller data volume
   - can be performed on the SE as part of post-job validation
   - utilities already exist for merging (eg. *hadd, eviocat,* etc.)

how to subset an evio file



how NOT to NOT subset an evio file

- ✘ Copy data on the SE prior to job submission
  - introduces extra step in processing
  - doubles load on the SE
  - *not necessary*
- ✘ Tell *hd_root* to skip forward *N* events before start of processing
  - increases total data transfer load by factor ~35
  - large cpu load increase from block unpacking
  - assumes we know in advance how many events per file
- ✘ Tell *hd_root* to skip forward *N* blocks before start of processing
  - still need to know when to stop
  - requires modification to the JANA framework
  - violates goal to run ***the same hd_root release on all production sites***

an *effective, light-weight* solution

customize the XRootD Posix preload client library

- no modifications to the Gluex software stack
- changes are local to the OSG singularity container
- correctness can be verified outside hd_root
- maintainable as a fork of the XRootD github repo

an *effective, light-weight* solution

what XRootD components needed to be touched?

- github fork rjones30 / xrootd from master on 11/12/2020
- updates enclosed in #ifdef EVIO_BLOCK_SUBSET_EXTENSION
  - src/XrdPosix/XrdPosixFile.hh (40 added lines)
  - src/XrdPosix/XrdPosixFile.cc (37 added lines)
  - src/XrdPosix/XrdPosixXrootd.cc (237 added lines)

an *effective, light-weight* solution

what changed?

- all existing preload library functionality retained
- added: recognition of a specially formatted xrootd url at open

To access the full input file, use the standard xrootd url, *eg. (actually exists!)*

`root://cn440.storrs.hpc.uconn.edu/gluex/rawdata/Run071728/hd_rawdata_071728_000.evio`

To get a subset with just 2 evio blocks + BOR + term, *eg. (actually exists!)*

`root://cn440.storrs.hpc.uconn.edu/gluex/rawdata/Run071728/hd_rawdata_071728_000.evio`**`:14,16`**

an *effective, light-weight* solution

what POSIX ops are modified for a subset file?

- read, lseek
- streampos values recomputed based on the virtual file sequence
- assumes that the file is evio-structured, not based on file extension.

To access the full input file, use the standard xrootd url, *eg. (actually exists!)*

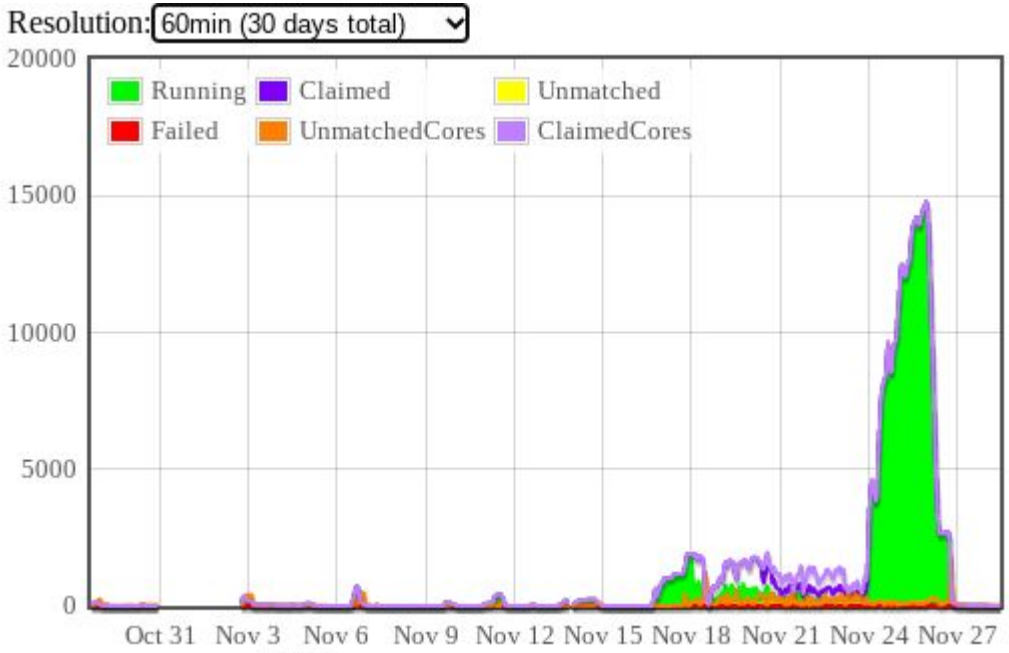
root://cn440.storrs.hpc.uconn.edu/gluex/rawdata/Run071728/hd_rawdata_071728_000.evio

To get a subset with just 2 evio blocks + BOR + term, *eg. (actually exists!)*

root://cn440.storrs.hpc.uconn.edu/gluex/rawdata/Run071728/hd_rawdata_071728_000.evio **:14,16**

a feasibility study

- 10 runs 71728 - 717423
  - 1880 raw evio files
  - 600M events
  - 35 TB
  - ~500k core-hr
- maps onto osg workflow
  - 4 cpu-hr / 1-core job
  - 68 jobs / evio file
  - 126,000 jobs
- production highlights
  - completed in 3 days, see plot
  - 15,000 running cores, max!



overlapping remote data access and processing

```
#!/bin/bash
# osgprod_worker.bash

function staging() {
...
}

mkfifo waitin
mkfifo waitout
staging & readloop pid=$!
hd_root [options]     waitin inblock.evio waitout \
                  waitin inblock.evio waitout \
                  waitin inblock.evio waitout \
                  waitin inblock.evio waitout \
                  waitin inblock.evio waitout
exit $?
```

overlapping remote data access and processing

```
function staging() {
    n=0
    for remote_input in $input_eviofile_list; do
        cat $remote_input >staging.evio                    # fetch the next block
        [ $n -gt 0 ] && cat /dev/null > waitout            # release the last block
        mv staging.evio inblock.evio                       # insert the next block
        cat /dev/null >waitin                              # release the next block
        n=`expr $n + 1`
    done
    cat /dev/null >waitout                                 # let hd_root finish
    rm waitin waitout inblock.evio                         # clean up and exit
}
```

Results

- SE - dcache with 750TB of space for input + output
- using dcache xrootd door
  - 6 instances: cn440...445, but only needed one
  - door functions only to forward connections to “movers” on “pool” hosts
  - input + output storage spread equally over 38 hosts
  - internal network 100Gb/s infiniband
  - external network 10Gb/s ethernet
- at peak, load remained reasonable (50 simulataneous transfers, 2 min. max)
- no long-lived sockets, all connections fetch just one block and disconnect
- no problems observed with transaction rates, all good!